Configuración, sintonización y optimización

Remo Suppi Boldrito

P07/M2103/02289



Índex

Introducción

Un aspecto fundamental, una vez que el sistema está instalado, es la configuración y adaptación del sistema a las necesidades del usuario y que las prestaciones del sistema sean lo más adecuadas posible a las necesidades que de él se demandan. GNU/Linux es un sistema operativo-eficiente que permite un grado de configuración excelente y una optimización muy delicada de acuerdo a las necesidades del usuario. Es por ello por lo que, una vez realizada una instalación (o en algunos casos una actualización), deben hacerse determinadas configuraciones vitales en el sistema. Si bien el sistema "funciona", es necesario efectuar algunos cambios (adaptación al entorno o sintonización) para permitir que estén cubiertas todas las necesidades del usuario/servicios que presta la máquina. Esta sintonización dependerá de dónde se encuentre funcionando la máquina, y se llevará a cabo, en algunos casos, para mejorar el rendimiento del sistema, y en otros (además), por cuestiones seguridad (ver el módulo 9, "Administrador de seguridad"). Cuando está en funcionamiento, es necesario monitorizar el sistema para ver su comportamiento y actuar en consecuencia. Si bien es un aspecto fundamental, la sintonización de un operativo muchas veces se relega a la opinión de expertos o *gurús* de la informática; pero conociendo los parámetros que afectan al rendimiento, es posible llegar a buenas soluciones haciendo un proceso cíclico de análisis, cambio de configuración, monitorización y ajustes.

1. Aspectos básicos

Antes de conocer cuáles son las técnicas de optimización, es necesario enumerar las causas que pueden afectar a las prestaciones de un sistema operativo [Maj96]. Entre éstas, se pueden mencionar:

a) Cuellos de botella en los recursos: la consecuencia es que todo el sistema irá más lento porque existen recursos que no pueden satisfacer la demanda a la que se les somete. El primer paso para optimizar el sistema es encontrar estos cuellos de botella y sus causas, conociendo sus limitaciones teóricas y prácticas.

b) Ley de Amdahl: según esta ley, “hay un límite de cuánto puede uno mejorar en velocidad una cosa si sólo se optimiza una parte de ella”; es decir, si se tiene un programa que utiliza el 10% de CPU y se optimiza reduciendo la utilización en un factor de 2, el programa mejorará sus prestaciones (*speedup*) en un 5%, lo cual puede significar un tremendo esfuerzo no compensado con los resultados.

c) Estimación del *speedup*: es necesario estimar cuánto mejorará para evitar esfuerzos y costes innecesarios. Se puede utilizar la ley anterior para valorar si es necesaria una inversión en tiempo o económica en el sistema.

d) Efecto burbuja: siempre se tiene la sensación de que, cuando se encuentra la solución a un problema, surge otro. Una manifestación de este problema es que el sistema se mueve constantemente entre problemas de CPU y problemas de entrada/salida, y viceversa.

e) Tiempo de repuesta frente a cantidad de trabajo: si se cuenta con veinte usuarios, mejorar en la productividad significará que todos tendrán más trabajo hecho al mismo tiempo, pero no mejores respuestas individualmente; podría ser que el tiempo de respuesta para algunos fuera mejor que para otros. Mejorar el tiempo de respuesta significa optimizar el sistema para que las tareas individuales tarden lo menos posible.

f) Psicología del usuario: dos parámetros son fundamentales: 1) el usuario estará insatisfecho generalmente cuando se produzcan variaciones en el tiempo de respuesta; y 2) el usuario no detectará mejoras en el tiempo de ejecución menores del 20%.

g) Efecto prueba: las medidas de monitorización afectan a las propias medidas. Se debe ir con cuidado cuando se realizan las pruebas por los efectos colaterales de los propios programas de medida.

h) Importancia de la media y la variación: se deben tener en cuenta los resultados, ya que si se obtiene una media de utilización de CPU del 50% cuando ha sido utilizada 100, 0, 0, 100, se podría llegar a conclusiones erróneas. Es importante ver la variación sobre la media.

Nota

Para optimizar hay que tener en cuenta la saturación de los recursos. Ley de Amdahl: relaciona los conocimientos de software y hardware disponible, el tiempo de respuesta y el número de trabajos.

i) Conocimientos básicos sobre el hardware del sistema por optimizar: para mejorar una cosa es necesario "conocer" si es susceptible de mejora. El encargado de la optimización deberá conocer básicamente el hardware subyacente (CPU, memorias, buses, caché, entrada/salida, discos, vídeo...) y su interconexión para poder determinar dónde están los problemas.

j) Conocimientos básicos sobre el sistema operativo por optimizar: del mismo modo que en el punto anterior, el usuario deberá conocer aspectos mínimos sobre el sistema operativo que pretende optimizar, entre los cuales se incluyen conceptos como procesos y *threads* (creación, ejecución, estados, prioridades, terminación), llamadas al sistema, *buffers* de caché, sistema de archivos, administración de memoria y memoria virtual (paginación, *swap*) y tablas del *kernel*.

1.1. Monitorización sobre UNIX System V

El /proc lo veremos como un directorio pero en realidad es un sistema de archivos ficticio, es decir, no existe sobre el disco y el *kernel* lo crea en memoria. Éste se utiliza para proveer de información sobre el sistema (originalmente sobre procesos, de aquí el nombre) que luego será utilizada por todos los comandos que veremos a continuación. A continuación veremos algunos archivos interesantes (consultar la página del manual para más información):

/proc/1: un directorio con la información del proceso 1 (el número del directorio es el PID del proceso)

/proc/cpuinfo: información sobre la CPU (tipo, marca, modelo, prestaciones...)

/proc/devices: lista de dispositivos configurados en el kernel.

/proc/dma: canales de DMA utilizados en ese momento

/proc/filesystems: sistemas de archivos configurados en el kernel.

/proc/interrupts: muestra cuales interrupciones están en uso y cuantas de ellas se han procesado.

/proc/ioports: ídem con los puertos

/proc/kcore: imagen de la memoria física del sistema.

/proc/kmsg: mensajes generados por el kernel que luego son enviados a syslog.

/proc/ksyms: Tabla de símbolos del kernel.

/proc/loadavg: carga del sistema

/proc/meminfo: información sobre la utilización de memoria.

/proc/modules: módulos cargados por el kernel.

/proc/net: información sobre los protocolos de red.

/proc/stat: estadísticas sobre el sistema.

/proc/uptime: desde cuando el sistema está funcionando.

/proc/version: versión del kernel.

Se debe tener en cuenta que estos archivos son visibles (texto) pero algunas veces los datos están en "crudo" y son necesarios comandos para interpretarlos, que serán los que veremos a continuación.

Los sistemas compatibles UNIX SV utilizan los comandos sar y sadc para obtener estadísticas del sistema (en FC incluidos dentro del paquete sysstat que incluye además iostat u mpstat). Su equivalente en GNU/Linux Debian es atsar (y atsadc), que es totalmente equivalente a los que hemos mencionado. El comando atsar lee contadores y estadísticas del fichero /proc y las muestra por la salida estándar. La primera forma de llamar al comando es:

```
atsar opciones t [n]n
```

donde muestra la actividad en *n* veces cada *t* segundos con una cabecera mostrando los contadores de actividad (el valor por defecto de *n* es 1). La segunda forma de llamarlo es:

```
atsar -opciones -s time -e time -i sec -f file -n day#
```

El comando extrae datos del archivo especificado por *-f* (por defecto /var/log/atsar/atsarxx, con *xx* el día del mes) y que fueron previamente guardados por atsadc (se utiliza para recoger los datos, salvarlos y procesarlos y en Debian está en /usr/lib/atsar). El parámetro *-n* puede ser utilizado para indicar el día del mes y *-s*, *-e* la hora de inicio-final, respectivamente. Para activar atsadc, por ejemplo, se podría incluir en /etc/cron.d/atsar una línea como la siguiente:

```
@reboot root test -x /usr/lib/atsadc && /usr/lib/atsar/atsadc /var/log/atsar/atsa'date +\%d'

10,20,30,40,50 * * * * root test -x /usr/lib/atsar/atsa1 && /usr/lib/atsar/atsa1
```

La 1.ª crea el archivo después de un reinicio. La 2.ª guarda los datos cada 10 minutos con el shell script atsa1, que llama al atsadc.

En atsar *(o* sar*)*, las opciones se utilizan para indicar qué contadores hay que mostrar; algunos de ellos son:

Opción	Descripción
u	Utilización CPU
d	Actividad de disco
I (i)	Número de interrupciones/s
v	Utilización de tablas en el *kernel*
y	Estadísticas de utilización de *ttys*
p	Información de paginación y actividad de *swap*
r	Memoria libre y ocupación de *swap*
l (L)	Estadísticas de red
L	Información de errores de red
w	Estadísticas de conexiones IP
t	Estadísticas de TCP
U	Estadísticas de UDP
m	Estadísticas de ICMP
N	Estadísticas de NFS
A	Todas las opciones

Nota

Monitorizar con atsar
- CPU: atsar -u
- Memoria: atsar -r
- Disco: atsar -d
- Paginación: atsar -p

Entre atsar y sar sólo existen algunas diferencias en cuanto a cómo muestran los datos y sar incluye unas cuantas opciones más (o diferentes). A continuación se verán algunos ejemplos de utilización del sar (exactamente igual que con atsar, sólo puede haber alguna diferencia en la visualización de los datos) y el significado de la información que genera:

1) Utilización de CPU

```
sar -u 4 5

Linux 2.6.19-prep (localhost.localdomain) 24/03/07
```

08:23:22	CPU	%user	%nice	%system	%iowait	%steal	%idle
08:23:26	all	0,25	0,00	0,50	0,00	0,00	99,25
08:23:30	all	0,00	0,00	0,00	0,00	0,00	100,00
08:23:34	all	0,00	0,00	0,00	0,00	0,00	100,00
08:23:38	all	0,25	0,00	0,00	0,00	0,00	99,75
08:23:42	all	0,00	0,00	0,00	0,00	0,00	100,00
Media:	all	0,10	0,00	0,10	0,00	0,00	99,80

- usr y system muestran el porcentaje de tiempo de CPU en el modo usuario con nice = 0 (normales) y en el modo *kernel*.

- nice, lo mismo, pero con procesos de usuario con nice > 0 (menor prioridad que la media).
- idle indica el tiempo no utilizado de CPU por los procesos en estado de espera (no incluye espera de disco).
- iowait es el tiempo que la CPU ha estado libre cuando el sistema estaba haciendo entrada o salida de disco.
- steal es el tiempo gastado inútilmente esperando por un CPU virtual (válido en entornos virtualizados).

En este caso `idle=100`, lo cual significa que la CPU está ociosa por lo que no hay procesos por ejecutar y la carga es baja; si `idle=10` y el número de procesos es elevado, debería pensarse en optimizar la CPU, ya que podría ser el cuello de botella del sistema.

2) Número de interrupciones por segundo

```
sar -I 4 5
```

```
Linux 2.6.19-prep (localhost.localdomain)            24/03/07
08:24:01 INTR intr/s
08:24:06 4 0,00
Media: 4 0,00
```

Muestra la información de la frecuencia de interrupciones de los niveles activos que se encuentran en /proc/interrupts. Nos es útil para ver si existe algún dispositivo que está interrumpiendo constantemente el trabajo de la CPU.

3) Memoria y *swap*

```
sar -r 4 5
```

```
Linux 2.6.19-prep (localhost.localdomain)            24/03/07
```

08:24:20 kbmemfree kbmemused %memused kbbuffers kbcached kbswpfree kbswpused %swpused kbswpcad

08:24:24 296516 729700 71,11 24260 459972 963860 0 0,00 0

08:24:28 296500 729716 71,11 24268 459968 963860 0 0,00 0

08:24:32 296516 729700 71,11 24268 459976 963860 0 0,00 0

08:24:36 296516 729700 71,11 24276 459976 963860 0 0,00 0

08:24:40 296500 729716 71,11 24276 459976 963860 0 0,00 0

Media: 296510 729706 71,11 24270 459974 963860 0 0,00 0

En este caso kbmemfree es la memoria principal (MP) libre; used la utilizada, *buffers* es la MP utilizada en *buffers*; cached es memoria principal utilizada en caché de páginas; swpfree/used el espacio libre/ocupado de *swap* . Es importante tener en cuenta que si no hay espacio en MP, las páginas de los procesos irán

a parar al *swap*, donde debe haber sitio. Esto se debe contrastar con la utilización de CPU. También se debe controlar que el tamaño de los *buffers* sea adecuado y esté en relación con los procesos que están realizando operaciones de entrada/salida.

Es también interesante el comando free (fc) que permite ver la cantidad de memoria en una visión simplificada:

```
  total used free shared buffers cached
Mem: 1026216 729716 296500 0 24324 459980
-/+ buffers/cache: 245412 780804
Swap: 963860 0 963860
```

Esto indica que de 1Gb casi ¾ de la memoria está ocupada y que casi ½ son de caché. Además nos indica que el *swap* no se está utilizando para nada, por lo que podemos concluir que el sistema está bien. Si quisiéramos más detalles, deberíamos utilizar el comando vmstat (o sar -r) para analizar qué es lo que está causando problemas o quién está consumiendo tanta memoria. A continuación se muestra una salida de vmstat 1 10:

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r b swpd free buff cache si so bi  bo    in   cs  us  sy   id  wa st
 0 0 0 295896 24384 459984 0 0 321  56    1249 724 11  2    81  5  0
 0 0 0 295896 24384 459984 0 0 0    28    1179 383 1   0    99  0  0
 0 0 0 295896 24384 460012 0 0 0    0     1260 498 0   0    100 0  0
 0 0 0 295896 24384 460012 0 0 0    0     1175 342     0    0   1000
 0 0 0 295896 24384 460012 0 0 0    0     1275 526 0   0    100 0  0
 1 0 0 295896 24392 460004 0 0 0    72    1176 356 0   0    99  1  0
 0 0 0 295896 24392 460012 0 0 0    0     1218 420 0   0    100 0  0
 0 0 0 295896 24392 460012 0 0 0    0     1216 436 0   0    100 0  0
 0 0 0 295896 24392 460012 0 0 0    0     1174 361 0   0    100 0  0
 1 0 0 295896 24392 460012 0 0 0    0     1260 492 0   0    100 0  0
```

4) Utilización de las tablas del *kernel*

```
sar -v 4 5
```

```
Linux 2.6.19-prep (localhost.localdomain)           24/03/07
08:24:48  dentunusd file-sz inode-sz super-sz %super-sz dquot-sz %dquot-sz rtsig-sz %rtsig-sz
08:24:52  19177     3904    15153    0         0,00      0         0,00    0           0,00
08:24:56  19177     3904    15153    0         0,00      0         0,00    0           0,00
08:25:00  19177     3904    15153    0         0,00      0         0,00    0           0,00
08:25:04  19177     3904    15153    0         0,00      0         0,00    0           0,00
08:25:08  19177     3904    15153    0         0,00      0         0,00    0           0,00
Media:    19177     3904    15153    0         0,00      0         0,00    0           0,00
```

En este caso, superb-sz es el número actual máximo de *superblocks* mantenido por el *kernel* para los sistemas de archivos montados; inode-sz, el número actual máximo de incore-inodes en el *kernel* necesario, es de uno por disco como mínimo; file-sz actual máximo número de archivos abiertos, dquota-sz actual máximo ocupación de entradas de cuotas (para las restantes opciones, consultar man sar (o atsar)). Esta monitorización se puede completar con el comando `ps -edaflm` (process status) y el comando top, que mostrarán la actividad y estado de los procesos en el sistema. A continuación, se muestran dos ejemplos de ambos comandos (sólo algunas líneas):

ps -edaflm

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	STIME	TTY	TIME	CMD
4	-	root	1	0	0	-	-	-	508	-	08:01	?	00:00:00	init [5]
1	-	root	1927	7	0	-	-	-	0	-	08:02	?	00:00:00	[kondemand/0]
1	-	rpc	2523	1	0	-	-	-	424	-	08:02	?	00:00:00	syslogd -m 0
5	S	rpc	2566	1	0	-	-	-	444	-	08:02	?	00:00:00	portmap
5	-	root	-	0	78	0	-	-	-	-	08:02	-	00:00:00	-
5		root	2587	1	0	-	-	-	472	-	08:02	?	00:00:00	rpc.statd
5	S	root	-	-	0	81		0	-	-	08:02	-	00:00:00	-
1	-	root	2620		1	0	-	-	1232	-	08:02	?	00:00:00	rpc.idmapd
1	S	root	-	-	0	75		0	-	defaul	08:02	-	00:00:00	-
5	-	root	2804	1	0	-	-	-	1294	-	08:02	?	00:00:00	/usr/sbin/sshd
5	S	root	-	-	0	84	0	-	-	-	08:02	-	00:00:00	-
5	-	root	2910	1	0	-	-	-	551	-	08:02	?	00:00:00	/usr/sbin/atd
5	S	root	-	-	0	84	0	-	-	-	08:02	-	00:00:00	-
4	-	root	3066	1	0	-	-	-	407	-	08:02	tty1	00:00:00	/sbin/mingetty tty1
4		root	3305	1	0	-	-	-	21636	-	08:03	?	00:00:01	nautilus --no-default-window --sm-
4	-	root	3305	1	0	-	-	-	21636	-	08:03	?	00:00:01	client-id default3
0	-	root	3643	3541	0	-	-	-	1123	-	08:17	pts/1	00:00:00	bash
4	-	root	3701	3643	0	-	-	-	1054	-	08:27	pts/1	00:00:00	ps -edaflm

Donde los parámetros reflejan el valor indicado en la variable del *kernel* para este proceso, los más importantes desde el punto de vista de la monitorización son: F flags (en este caso 1 es con superprivilegios, 4 creado desde el inicio *daemon*), S es el estado (D: no-interrumpible durmiendo entrada/salida, R: ejecutable o en cola, S: durmiendo, T: en traza o parado, Z: muerto en vida, 'zombie'). PRI es la prioridad; NI es nice; STIME, el tiempo de inicio de ejecución; TTY, desde donde se ha ejecutado; TIME, el tiempo de CPU; CMD, el programa que se ha ejecutado y sus parámetros. Si se quiere salida con refresco (configurable), se puede utilizar el comando top, que muestra unas estadísticas generales (procesos, estados, carga, etc.), y después, información de cada uno de ellos similar al ps, pero se actualiza cada 5 segundos por defecto:

Nota

Consultar el comando man ps o man top para una descripción de los parámetros y características

```
top - 08:26:52 up 25 min, 2 users, load average: 0.21, 0.25, 0.33
Tasks: 124 total, 1 running, 123 sleeping, 0 stopped, 0 zombie
Cpu(s): 10.8%us, 2.1%sy, 0.0%ni, 82.0%id, 4.9%wa, 0.1%hi, 0.1%si, 0.0%st
Mem: 1026216k total, 731056k used, 295160k free, 24464k buffers
Swap: 963860k total, 0k used, 963860k free, 460208k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3541	root	15	0	42148	14m	981	S	1.9	1.5	0:00.76	gnome-terminal
3695	root	15	0	260	944	1650	R	1.9	0.1	0:00.02	top
1	root	RT	0	2032	680	580	S	0.0	0.1	0:00.85	init
2	root	34	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
3	root	RT	19	0	0	0	S	0.0	0.0	0:00.04	ksoftirqd/0
4	root	10	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
5	root	16	-5	0	0	0	S	0.0	0.0	0:00.00	events/0
6	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khelper
7	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
53	root	11	-5	0	0	0	S	0.0	0.0	0:00.01	kblockd/0
54	root	15	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
177	root	18	-5	0	0	0	S	0.0	0.0	0:00.00	cqueue/0
178	root	18	-5	0	0	0	S	0.0	0.0	0:00.00	ksuspend_usbd
181	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khubd
183	root	10	-5	0	0	0	S	0.0	0.0	0:00.01	kseriod
203	root	23	0	0	0	0	S	0.0	0.0	0:00.00	pdflush
204	root	15	0	0	0	0	S	0.0	0.0	0:00.03	pdflush

Debian Linux también incluye todo un conjunto de herramientas de monitorización equivalentes a sar, pero que son originarias de UNIX BSD y que presentan una funcionalidad similar, aunque desde diferentes comandos: vmstat (estadísticas de CPU, memoria y entrada/ salida), iostat (estadísticas de discos y CPU), uptime (carga de CPU y estado general).

1.2. Optimizando el sistema

A continuación veremos algunas recomendaciones para optimizar el sistema en función de los datos obtenidos.

1) Resolver los problemas de memoria principal

Se debe procurar que la memoria principal pueda acoger un porcentaje elevado de procesos en ejecución, ya que, si no es así, el sistema operativo podrá paginar e ir al *swap*; pero esto significa que la ejecución de ese proceso se degradará notablemente. Si se agrega memoria, el tiempo de respuesta mejorará notablemente. Para ello, se debe tener en cuenta el tamaño de los procesos (SIZE) en estado *R* y agregarle la que utiliza el *kernel*, que se puede obtener con el comando dmesg, que nos mostrará (o con *free*), por ejemplo:

Memory:

255048k/262080k available (1423k kernel core, 6644k reserved, 466k data, 240k init, Ok highmem

Luego se deberá comparar con la memoria física y analizar si el sistema está limitado por la memoria (se verá con atsar -r y -p mucha actividad de paginación).

Las soluciones para la memoria son obvias: o se incrementa la capacidad o se reducen las necesidades. Por el coste actual de la memoria, es más adecuado incrementar su tamaño que emplear muchas horas para ganar un centenar de bytes por quitar, ordenar o reducir requisitos de los procesos en su ejecución. Reducir los requisitos puede hacerse reduciendo las tablas del *kernel*, quitando módulos, limitando el número máximo de usuarios, reduciendo los *buffers*, etc.; todo lo cual degradará el sistema (efecto burbuja) y las prestaciones serán peores (en algunos casos, el sistema puede quedar totalmente no operativo).

Nota

Dónde mirar?
1.º Memoria
2.º CPU
3.º Entrada/salida
4.º TCP/IP
5.º Kernel

Otro aspecto que se puede reducir es la cantidad de memoria de los usuarios eliminando procesos redundantes y cambiando la carga de trabajo. Para ello, se deberán monitorizar los procesos que están durmiendo (*zombies*) y eliminarlos, o aquellos que no progresan en su entrada/salida (saber si son procesos activos, cuánto CPU llevan gastado y si los 'usuarios están por ellos'). Cambiar la carga de trabajo es utilizar planificación de colas para que los procesos que necesitan gran cantidad de memoria se puedan ejecutar en horas de poca actividad (por ejemplo, por la noche lanzándolos con el comando at).

2) Mucha utilización de CPU

Básicamente nos la da el tiempo idle (valores bajos). Con ps o top se deben analizar qué procesos son los que 'devoran CPU' y tomar decisiones como: posponer su ejecución, pararlos temporalmente, cambiar la prioridad (menos conflictivo de todos, se puede utilizar el comando renice prioridad PID), optimizar el programa (para la próxima vez) o cambiar la CPU (o agregar otra). Como ya se ha mencionado, GNU/Linux utiliza el directorio /proc para man-

tener todas las variables de configuración del *kernel* que pueden ser analizadas y, en cierto caso, 'ajustadas', para lograr prestaciones diferentes o mejores.

Para ello, se debe utilizar el comando `systune dump > /tmp/sysfile` para obtener todas las variables y sus valores en el archivo /tmp/sysfile (en otras distribuciones se puede hacer con sysctl). Este archivo se puede editar, cambiar la variable correspondiente y luego utilizar el comando `systune -c /tmp/sysfile` para cargarlas nuevamente en el /proc. El comando systune también lee por defecto si no tiene la opción -c de /etc/systune.conf. En este caso, por ejemplo, se podría modificar (procédase con cuidado, porque el *kernel* puede quedar fuera de servicio) las variables de la categoría /proc/sys/vm (memoria virtual) o /proc/sys/kernel (configuración del *core* del *kernel*).

En este mismo sentido, también (para expertos o desesperados) se puede cambiar el tiempo máximo (*slice*) que el administrador de CPU (*scheduler*) del sistema operativo dedica a cada proceso en forma circular (si bien es aconsejable utilizar *renice* como práctica). Pero GNU/Linux, a diferencia de otros operativos, es un valor fijo dentro del código, ya que está optimizado para diferentes funcionalidades (pero es posible tocarlo). Se puede "jugar" (a su propio riesgo) con un conjunto de variables que permiten tocar el *time slice* de asignación de CPU (kernel-source-2.x.x/kernel/sched.c).

3) Reducir el número de llamadas

Otra práctica adecuada para mejorar las prestaciones es reducir el número de llamadas al sistema de mayor coste en tiempo de CPU. Estas llamadas son las invocadas (generalmente) por el *shell* fork() y exec(). Una configuración inadecuada de la variable PATH y debido a que la llamada exec() no guarda nada en caché, el directorio actual (indicado por ./), puede tener una relación desfavorable de ejecución. Para ello, siempre habrá que configurar la variable PATH con el directorio actual como última ruta. Por ejemplo, en *bash* (o en .bashrc) hacer: export PATH = $PATH. Si no es así, no está el directorio actual, o si está, rehacer la variable PATH para ponerlo como última ruta.

Se debe tener en cuenta que una alta actividad de interrupciones puede afectar a las prestaciones de la CPU con relación a los procesos que ejecuta. Mediante monitorización (atsar -I) se puede mirar cuál es la relación de interrupciones por segundo y tomar decisiones con respecto a los dispositivos que las causan. Por ejemplo, cambiar de módem por otro más inteligente o cambiar la estructura de comunicaciones si detectamos una actividad elevada sobre el puerto serie donde se encuentra conectado.

4) Mucha utilización de disco

Después de la memoria, un tiempo de respuesta bajo puede ser debido al sistema de discos. En primer lugar se debe verificar que se disponga de tiempo de CPU (por ejemplo, idle > 20%) y que el número de entrada/salida sea ele-

vado (por ejemplo, > 30 entrada/salida/s) utilizando atsar -u y atsar -d. Las soluciones pasan por:

a) En un sistema multidisco, planificar dónde se encontrarán los archivos más utilizados para equilibrar el tráfico hacia ellos (por ejemplo /home en un disco y /usr sobre otro) y que puedan utilizar todas las capacidades de la entrada/salida con caché y concurrente de GNU/Linux (incluso, por ejemplo, planificar sobre qué bus ide se colocan). Comprobar luego que existe un equilibrio del tráfico con atsar -d (o con iostat). En situaciones críticas se puede considerar la compra de un sistema de discos RAID que realizan este ajuste de forma automática.

b) Tener en cuenta que se obtienen mejores prestaciones sobre dos discos pequeños que sobre uno grande del tamaño de los dos anteriores.

c) En sistemas con un solo disco, generalmente se realizan, desde el punto de vista del espacio, cuatro particiones de la siguiente manera (desde fuera hacia dentro): /, swap, /usr, /home; pero que genera pésimas respuestas de entrada/salida porque si, por ejemplo, un usuario compila desde su directorio /home/user y el compilador se encuentra en /usr/bin, la cabeza del disco se moverá por toda su longitud. En este caso es mejor unir las particiones /usr y /home en una sola (más grande) aunque puede representar algunos inconvenientes en cuanto a mantenimiento.

d) Incrementar los *buffers* de caché de la entrada/salida (ver, por ejemplo: /proc/ide/hd...).

e) Si se utiliza un ext2fs, se puede utilizar el comando: dumpe2fs -h /dev/hd... para obtener información sobre el disco y tune2fs /dev/hd... para cambiar algunos de los parámetros configurables del disco.

f) Obviamente, el cambio del disco por uno de mayor velocidad (RPM) siempre tendrá un impacto positivo en un sistema limitado por entrada/salida de disco. [Maj96]

5) Mejorar aspectos de TCP/IP

k) Examinar la red con el comando atsar (o también con `netstat -i` o con `netstat -s | more`) para analizar si existen paquetes fragmentados, errores, *drops*, *overflows*, etc., que puedan estar afectando a las comunicaciones y con ello al sistema (por ejemplo, en un servidor de NFS, NIS, Ftp o Web). Si se detectan problemas, se debe analizar la red para considerar las siguientes actuaciones:

a) Fragmentar la red a través de elementos activos que descarten paquetes con problemas o que no son para máquinas del segmento.

b) Planificar dónde estarán los servidores para reducir el tráfico hacia ellos y los tiempos de acceso.

c) Ajustar parámetros del *kernel* (/proc/sys/net/), por ejemplo, para obtener mejoras en el *throughput* hacer:

`echo 600 > /proc/sys/net/core/netdev` max backlog (por defecto 300).

6) Otras acciones sobre parámetros del *kernel*

Existe otro conjunto de parámetros sobre el *kernel* que es posible sintonizar para obtener mejores prestaciones, si bien, teniendo en cuenta lo que hemos tratado anteriormente, se debe ir con cuidado, ya que podríamos causar el efecto contrario o inutilizar el sistema. Consultar en la distribución del código fuente en *kernel*- *source*-2.4.18/Documentation/sysctl los archivos vm.txt, fs.txt, kernel.txt y sunrpc.txt:

a) /proc/sys/vm: controla la memoria virtual (MV) del sistema. La memoria virtual permite que los procesos que no entran en memoria principal sean aceptados por el sistema pero en el dispositivo de swap, por lo cual, el programador no tiene límite para el tamaño de su programa (obviamente debe ser menor que el dispositivo de *swap*). Los parámetros susceptibles de sintonizar se pueden cambiar muy fácilmente con *gpowertweak*.

b) /proc/sys/fs: se pueden ajustar parámetros de la interacción kernel-FS tal como file-max.

c) Y también sobre /proc/sys/kernel, /proc/sys/sunrpc

7) Generar el kernel adecuado a nuestras necesidades

La optimización del *kernel* significa escoger los parámetros de compilación de acuerdo a nuestras necesidades. Es muy importante primero leer el archivo readme de /usr/src/linux. Una buena configuración del *kernel* permitirá que éste se ejecute más rápido, que se disponga de más memoria para los procesos de usuario y además resultará más estable. Hay dos formas de construir un *kernel*: monolítico (mejores prestaciones), o modular (basado en módulos tendrá mejor portabilidad si tenemos un sistema muy heterogéneo y no se desea compilar un *kernel* para cada uno de ellos). Para compilar su propio *kernel* y adaptarlos a su hardware y necesidades, cada distribución tiene sus reglas (si bien el procedimiento es similar).

8) Es interesante consultar los siguientes artículos:

http://people.redhat.com/alikins/system_tuning.html sobre información de optimización de sistemas servidores Linux.

http://www.linuxjournal.com/article.php?sid=2396 Performance Monitoring Tools for Linux, si bien es un artículo antiguo y algunas opciones no están disponibles, la metodología es totalmente vigente.

1.3. Optimizaciones de carácter general

Existen una serie de optimizaciones de índole general que pueden mejorar las prestaciones del sistema:

1) Bibliotecas estáticas o dinámicas: cuando se compila un programa, se puede hacer con una biblioteca estática (libr.a), cuyo código de función se incluye en el ejecutable o con una dinámica (libr.so.xx.x), donde se carga la biblioteca en el momento de la ejecución. Si bien las primeras garantizan código portable y seguro, consumen más memoria. El programador deberá decidir cuál es la adecuada para su programa incluyendo -static en las opciones del compilador (no ponerlo significa dinámicas) o --disable-shared, cuando se utiliza el comando configure. Es recomendable utilizar (casi todas las distribuciones nuevas lo hacen) la biblioteca estándar libc.a y libc.so de versiones 2.2.x o superiores (conocida como Libc6) que reemplaza a las anteriores.

2) Selección del procesador adecuado: generar código ejecutable para la arquitectura sobre la cual correrán las aplicaciones. Algunos de los parámetros más influyentes del compilador son: -march (por ejemplo, marchi 686 o -march k6) haciendo simplemente gcc -marchi 686, el atributo de optimización -O1,2,3 (-O3 generará la versión más rápida del programa, gcc -O3 -march = i686) y los atributos -f (consultar la documentación para los diferentes tipos).

3) Optimización del disco: en la actualidad, la mayoría de ordenadores incluye disco UltraDMA (100) por defecto; sin embargo, en una gran cantidad de casos no están optimizados para extraer las mejores prestaciones. Existe una herramienta (`hdparm`) que permite sintonizar el *kernel* a los parámetros del disco tipo IDE. Se debe tener cuidado con esta utilidad, sobre todo en discos UltraDMA (verificar en el BIOS que los parámetros para soporte por DMA están habilitados), ya que pueden inutilizar el disco. Consultar las referencias y la documentación ([Mou01] y `man hdparm`) sobre cuáles son (y el riesgo que comporta) las optimizaciones más importantes, por ejemplo: -c3, -d1, -X34, -X66, -X12, -X68, -mXX, -a16, -u1, -W1, -k1, -K1. Cada opción significa una optimización y algunas son de altísimo riesgo, por lo que habrá que conocer muy bien el disco. Para consultar los parámetros optimizados, se podría utilizar hdparm -vtT /dev/hdX (donde X es el disco optimizado) y la llamada a hdparm con todos los parámetros se puede poner en /etc/init.d para cargarla en el *boot*.

1.4. Configuraciones complementarias

Existen más configuraciones complementarias desde el punto de vista de la seguridad que de la optimización, pero son necesarias sobre todo cuando el sistema está conectado a una intranet o a Internet. Estas configuraciones implican las siguientes acciones [Mou01]:

a) Impedir que se pueda arrancar otro sistema operativo: si alguien tiene acceso físico a la máquina, podría arrancar con otro sistema operativo preconfigurado y modificar el actual, por lo que se debe inhibir desde el BIOS del ordenador el *boot* por *floppy* o CD-ROM y poner un *password* de acceso (recordar el *password* del BIOS, ya que, de otro modo, podría causar problemas cuando se quisiera cambiar la configuración).

b) Configuración y red: es recomendable desconectar la red siempre que se deseen hacer ajustes en el sistema. Se puede quitar el cable o deshabilitar el dispositivo con `/etc/init.d/networking stop` (*start* para activarla de nuevo) o con `ifdown eth0` (`ifup eth0` para habilitarla) para un dispositivo en concreto.

c) Modificar los archivos de /etc/security, de acuerdo a las necesidades de utilización y seguridad del sistema. Por ejemplo, en access.conf sobre quién puede hacer un *login* al sistema.

```
Formato:

permisssion:   users   : origins
+o -           : usuarios : desde donde
-:ALL EXCEPT root: tty1                    Impide el acceso a todos
                                           no-root sobre tty1.
-:ALL EXCEPT user1 user2 user3:console     Impide el acceso excepto
                                           user1,2,3 pero el último
                                           sólo desde consola.
-:user1:ALL EXCEPT LOCAL .uoc.edu 'group.conf':
```

También se debería por ejemplo configurar los grupos para controlar qué y cómo y también los límites máximos (limits.conf) para establecer los tiempos máximos de utilización de CPU, E/S, etc. para evitar por ejemplo DoS.

d) Mantener la seguridad del *password* de *root*: utilizar como mínimo 6 caracteres, con uno, por lo menos, en mayúsculas o algún carácter '.-_,', que sea no trivial; asimismo, es recomendable activar el envejecimiento para forzar a cambiarlo periódicamente, así como limitar el número de veces con *password* incorrecto. Asimismo, habrá que cambiar el parámetro min*x* la entrada en /etc/pam.d/passwd para indicar el número mínimo de caracteres que se utilizarán en el *password* (*x* es el número de caracteres).

e) No acceder al sistema como root: crear una cuenta como sysadm y trabajar con ella. Si se accede remotamente, habrá siempre que utilizar shh para conectarse al sysadm y, en caso de ser necesario, realizar un *su* - para trabajar como *root*.

f) Tiempo máximo de inactividad: inicializar la variable TMOUT, por ejemplo a 360 (valor expresado en segundos), que será el tiempo máximo de inactividad que esperará el *shell* antes de bloquearse; se puede poner en los archivos de configuración del *shell* (por ejemplo, /etc/profile, /.bashrc...). En caso de utilizar entornos gráficos (KDE, Gnome, etc.), activar el salvapantallas con *password*.

g) Configuración del NFS en forma restrictiva: en el /etc/exports exportar sólo lo necesario, no utilizar comodines (*wildcards*), permitir sólo el acceso

de lectura y no permitir el acceso de escritura por *root*, por ejemplo, con */directorio_exportado host.domain.com* (*ro*, *root_squash*).

h) Evitar arranques desde el lilo (o grub) con parámetros: se puede iniciar el sistema como linux single, lo cual arrancará el sistema operativo en modo de usuario único. Configurar el sistema para que el arranque de este modo siempre sea con *password*. Para ello, en el archivo /etc/inittab verificar que existe la línea: S:wait:/sbin/sulogin y que tiene habilitado el /bin/sulogin. Además, el archivo /etc/ lilo.conf debe tener los permisos adecuados para que nadie lo pueda modificar excepto el *root* (chmod 600 /etc/lilo.conf). Para prevenir cambios accidentales, cambiar el atributo de bloqueo con `chattr +i /etc/lilo.conf` (utilizar -i cuando se desee cambiar). Este archivo permite una serie de opciones que es conveniente considerar: timeout o, si el sistema tiene sólo un sistema operativo para que haga el *boot* inmediatamente, *restricted*, para evitar que puedan insertar comandos en el momento del *boot* como linux init = /bin/sh, y tener acceso como *root* sin autorización; en este caso, debe ir acompañado de password palabra-de-password; si sólo se pone *password*, solicitará el *password* para cargar la imagen del *kernel*. Grub tiene opciones similares.

i) Control de la combinación Ctrl-Alt-Delete. Para evitar que se pueda apagar la máquina desde el teclado, insertar un comentario (#) en la primera columna de la línea siguiente:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
del archivo /etc/inittab. Activar los cambios con telinit q.
```

j) Evitar servicios no ofrecidos: bloquear el archivo /etc/services para no admitir servicios no contemplados bloqueando el archivo con *chattr +i /etc/services*.

k) Conexión del *root:* modificar el archivo /etc/securetty que contiene las TTY y VC (*virtual console*) en que se puede conectar el *root* dejando sólo una de cada, por ejemplo, *tty1* y *vc*/1, y si es necesario conectarse como sysadm y hacer un su.

l) Eliminar usuarios no utilizados: borrar los usuarios/grupos que no sean necesarios, incluidos los que vienen por defecto (por ejemplo, *operator, shutdown, ftp, uucp, games*...), y dejar sólo los necesarios (*root, bin, daemon, sync, nobody, sysadm*) y los que se hayan creado con la instalación de paquetes o por comandos (lo mismo con /etc/group). Si el sistema es crítico, podría considerarse bloquear (`chattr +i file`) los archivos /etc/passwd, /etc/shadow, /etc/group, /etc/gsahdow para evitar su modificación (cuidado con esta acción, porque no permitirá cambiar posteriormente los passwd).

m) Montar las particiones en forma restrictiva: utilizar en /etc/fstab atributos para las particiones tales como nosuid (no permite suplantar al usuario o grupo sobre la partición), nodev (no interpreta dispositivos de caracteres o bloques sobre esta partición) y noexec (no permite la ejecución de archivos sobre esta partición). Por ejemplo:

```
/tmp /tmp ext2 defaults,nosuid,noexec 0 0
```

También es aconsejable montar el /boot en una partición separada y con atributos ro.

n) Protecciones varias: cambiar a 700 las protecciones de los archivos de /etc/init.d (servicios del sistema) para que sólo el *root* pueda modificarlos, arrancarlos o pararlos y modificar los archivos /etc/issue y /etc/issue.net para que no den información (sistema operativo, versión...) cuando alguien se conecta por telnet, ssh, etc.

o) SUID y SGID: un usuario podrá ejecutar como propietario un comando si tiene el bit SUID o SGID activado, lo cual se refleja como una 's' SUID (-rwsr-xr-x) y SGID (-r-xr-sr-x). Por lo tanto, es necesario quitar el bit (chmod a-s file) a los comandos que no lo necesitan. Estos archivos pueden ser buscados con:

```
find / -type f -perm -4000 o -perm -2000 –print
```

Se debe proceder con cuidado respecto a los archivos que quita el SUID- GUID porque el comando podría quedar inútil.

p) Archivos sospechosos: buscar periódicamente archivos con nombres no usuales, ocultos o sin un uid/gid válido, como '...' (tres puntos), '.. ' (punto punto espacio), '..^G', para ello, habrá que utilizar:

```
find / -name ".*" -print | cat -v
```

o si no

```
find / name ".." -print
```

Para buscar uid/gid no válidos, utilizar: `find / -nouser o -nogroup` (cuidado, porque algunas instalaciones se hacen con un usuario que luego no está definido y que el administrador debe cambiar).

q) Conexión sin *password*: no permitir el archivo .rhosts en ningún usuario a no ser que sea estrictamente necesario (se recomienda utilizar ssh con clave pública en lugar de métodos basados en .rhosts).

r) X Display manager: modificar el archivo /etc/X11/xdm/Xaccess para indicar los *hosts* que se podrán conectar a través de XDM y evitar que cualquier *host* pueda tener una pantalla de *login*.

1.5. Monitorización

Existen dos herramientas muy interesantes para la monitorización del sistema: Munin y Monit. Munin produce gráficos sobre diferentes parámetros del

servidor (load average, memory usage, CPU usage, MySQL throughput, eth0 traffic, etc.) sin excesivas configuraciones, mientras que monit verifica la disponibilidad de servicios tales como Apache, MySQL, Postfix, y toma diferentes acciones como por ejemplo reactivarlo si el servicio no está presente. La combinación da gráficos importantes para reconocer dónde y qué está generando problemas.

Consideremos que nuestro sistema se llama pirulo.org y tenemos nuestra página configurada como www.pirulo.org con los documentos en /var/www/pirulo.org/web. Para instalar Munin sobre Debian Sarge, hacemos por ejemplo `apt-get install munin munin-node`.

Luego debemos configurar munin (/etc/munin/munin.conf) con:

```
dbdir /var/lib/munin
htmldir /var/www/www.pirulo.org/web/monitoring
logdir /var/log/munin
rundir /var/run/munin
tmpldir /etc/munin/templates
[pirulo.org]
 address 127.0.0.1
 use_node_name yes
```

A continuación se crea el directorio, se cambian los permisos y se reinicia el servicio.

```
mkdir -p /var/www/pirulo.org/web/monitoring
chown munin:munin /var/www/pirulo.org/web/monitoring
/etc/init.d/munin-node restart
```

Después de unos minutos se podrán ver los primeros resultados en http://www.pirulo.org/monitoring/ en el navegador. Por ejemplo dos gráficas (carga y memoria) se muestran a continuación.

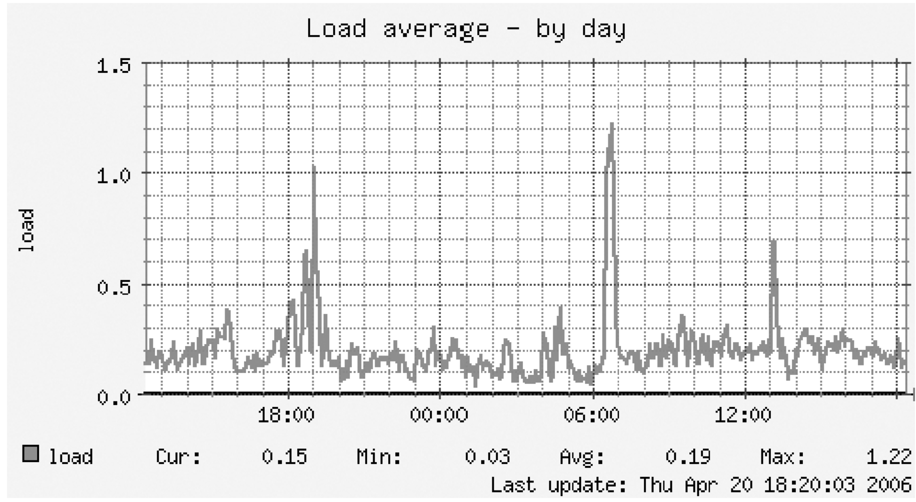


Figura 1

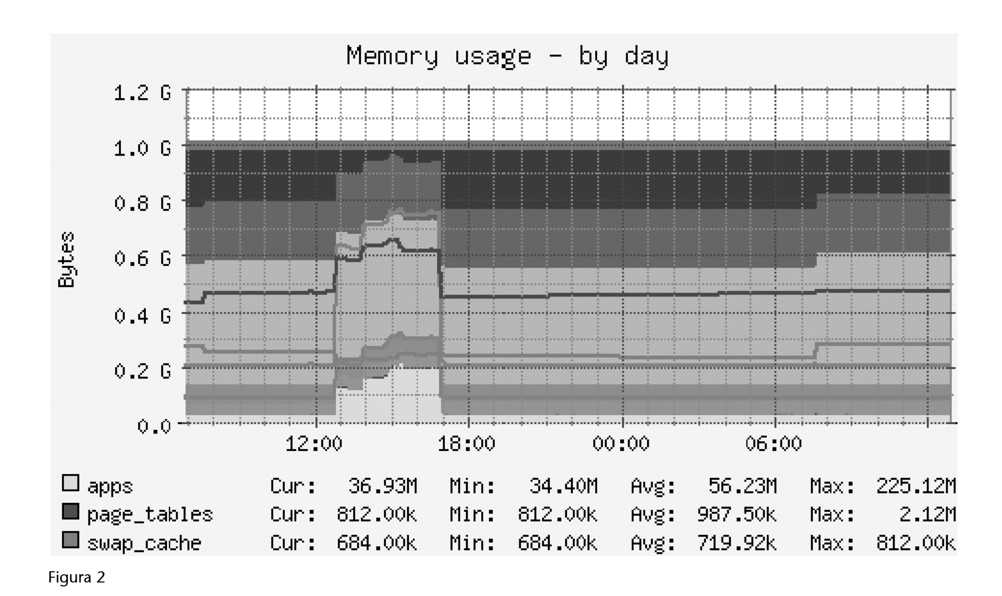


Figura 2

Si se quiere mantener la privacidad de las gráficas, basta con poner un passwd al acceso con apache al directorio. Por ejemplo, se pone en el directorio /var/www/pirulo.org/web/monitoring el archivo .htaccess con el siguiente contenido:

```
AuthType Basic
AuthName "Members Only"
AuthUserFile /var/www/pirulo.org/.htpasswd
<limit GET PUT POST>
require valid-user
</limit>
```

Luego se debe crear el archivo de passwd en /var/www/pirulo.org/.htpasswd con el comando (como *root*):

```
htpasswd -c /var/www/pirulo.org/.htpasswd admin
```

Cuando nos conectemos al www.pirulo.org/monitoring, no pedirá el usuario (admin) y el passwd que hemos introducido después del comando anterior.

Para instalar monit, hacemos `apt-get install` monit y editamos /etc/monit/monitrc. El archivo por defecto incluye un conjunto de ejemplos, pero se pueden obtener más desde http://www.tildeslash.com/monit/doc/examples.php. Si queremos, por ejemplo, monitorizar proftpd, sshd, mysql, apache, y postfix, habilitando la interfaz web de monit sobre el puerto 3333 podemos hacer sobre monitrc:

```
set daemon 60
set logfile syslog facility log_daemon
set mailserver localhost
set mail-format { from: monit@pirulo.org }
set alert root@localhost
set httpd port 3333 and
 allow admin:test
```

```
check process proftpd with pidfile /var/run/proftpd.pid
 start program = "/etc/init.d/proftpd start"
 stop program = "/etc/init.d/proftpd stop"
 if failed port 21 protocol ftp then restart
 if 5 restarts within 5 cycles then timeout

check process sshd with pidfile /var/run/sshd.pid
 start program "/etc/init.d/ssh start"
 stop program "/etc/init.d/ssh stop"
 if failed port 22 protocol ssh then restart
 if 5 restarts within 5 cycles then timeout

check process mysql with pidfile /var/run/mysqld/mysqld.pid
 group database
 start program = "/etc/init.d/mysql start"
 stop program = "/etc/init.d/mysql stop"
 if failed host 127.0.0.1 port 3306 then restart
 if 5 restarts within 5 cycles then timeout

check process apache with pidfile /var/run/apache2.pid
 group www
 start program = "/etc/init.d/apache2 start"
 stop program = "/etc/init.d/apache2 stop"
 if failed host www.pirulo.org port 80 protocol http
 and request "/monit/token" then restart
 if cpu is greater than 60% for 2 cycles then alert
 if cpu > 80% for 5 cycles then restart
 if totalmem > 500 MB for 5 cycles then restart
 if children > 250 then restart
 if loadavg(5min) greater than 10 for 8 cycles then stop
 if 3 restarts within 5 cycles then timeout

check process postfix with pidfile /var/spool/postfix/pid/master.pid
 group mail
 start program = "/etc/init.d/postfix start"
 stop program = "/etc/init.d/postfix stop"
 if failed port 25 protocol smtp then restart
 if 5 restarts within 5 cycles then timeout
```

Consultar el manual para más detalles http://www.tildeslash.com/monit/doc/manual.php. Para verificar que el servidor Apache funciona Monit, hemos puesto en la configuración que acceda a *if failed host www.pirulo.org port 80 protocol http and request "/monit/token" then restart*. Si no puede acceder, significa que Apache no funciona, por lo cual este archivo deberá existir (mkdir /var/www/pirulo.org/web/monit; echo "pirulo" > /var/www/pirulo.org/web/monit/token). También se puede configurar monit para que funcione sobre SSL (ver http://www.howtoforge.com/server_monitoring_monit_munin_p2).

Por último se debe modificar /etc/default/monit para habilitar a monit y cambiar startup=1 y CHECK_INTERVALS=60 por ejemplo (en segundos). Si ponemos en marcha monit (`/etc/init.d/monit start`) y nos conectamos http://www.pirulo.org:3333, se verá alguna pantalla similar a:

Monit Service Manager

Process	Status	Uptime	CPU	Memory
proftpd	runnig	1h 19m	0,0%	1,2% [2348 Kb]
sshd	runnig	1h 56m	0,0%	0,7% [1508 Kb]
mysql	runnig	1h 29m	0,0%	7,1% [13664Kb]
apache	runnig	15m	0,0%	5,0% [9628 Kb]
postfix	runnig	1h 26m	0,0%	0,6% [1322 Kb]

Process status

Parameter	Value
Name	apache
Pid file	/var/runapache2.pid
Status	running
Group	www
Monitoring mode	active
Monitoring status	monitored
Start program	/etc/nt.d/apache2 start
Stop program	/etc/nt.d/apache2 stop

Figura 3

Existen herramientas más sofisticadas para la monitorización de red y servicios de red utilizando SNMP (simple network management protocol) y MRTG (multi-router traffic grapher) por ejemplo. Más información al respecto se puede encontrar en http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch22_:_Monitoring_Server_Performance.

El MRTG (http://oss.oetiker.ch/mrtg/) fue creado básicamente para graficar datos de red, pero se pueden utilizar otros datos para visualizar su comportamiento, por ejemplo, para generar las estadísticas de carga (load average) del servidor. Para ello hacemos uso de los paquetes mrtg y atsar. Una vez instalados, configuramos el fichero /etc/mrtg.cfg:

```
WorkDir: /var/www/mrtg
Target[average]: `/usr/local/bin/cpu-load/average`
MaxBytes[average]: 1000
Options[average]: gauge, nopercent, growright, integer
YLegend[average]: Load average
kMG[average]: ,,
ShortLegend[average]:
Legend1[average]: Load average x 100
LegendI[average]: load:
LegendO[average]:
Title[average]: Load average x 100 for pirulo.org
PageTop[average]: <H1>Load average x 100 for pirulo.org</H1>
 <TABLE>
 <TR><TD>System:</TD>
 <TD>pirulo.org</TD></TR>
 <TR><TD>Maintainer:</TD> <TD>webmaster@pirulo.org</TD></TR>
 <TR><TD>Max used:</TD> <TD>1000</TD></TR>
 </TABLE>
```

Para generar los datos con `atsar` (o sar) creamos un script en /usr/local/bin/cpu-load/average (que tenga permisos de ejecución para todos) que pasará los datos a mrtg:

```
#!/bin/sh
load=`/usr/bin/atsar -u 1 | tail -n 1 | awk -F" " '{print $10}'`
echo "$load * 100" | bc | awk -F"." '{print $1}'
```

Deberemos crear y cambiar los permisos del directorio /var/www/mrtg. Por defecto, mrtg se ejecuta en el cron, pero si después queremos ejecutarlo, podemos hacer mrtg /etc/mrtg.cfg y esto generará las gráficas en /var/www/mrtg/ average.html que podremos visualizar con un navegador desde http:// www.pirulo.org/mrtg/averange.html.

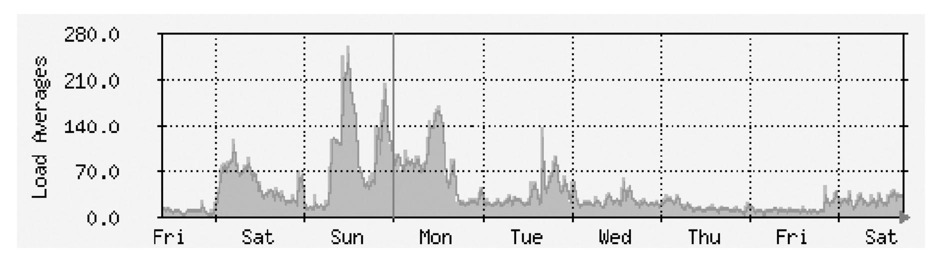


Figura 4

Otros paquetes interesantes a tener en cuenta para monitorizar un sistema son:

- Frysk (http://sources.redhat.com/frysk/): el objetivo del proyecto frysk es crear un sistema de monitorización distribuido e inteligente para monitorizar procesos y threads.

- Cacti (http://cacti.net/): Cacti es un solución gráfica diseñada para trabajar conjuntamente con datos de RRDTool's. Cacti provee diferentes formas de gráficas, métodos de adquisición y características que puede controlar el usuario muy fácilmente y es una solución que se adapta desde una máquina a un entorno complejo de máquinas, redes y servidores.

A continuación se describirán otras herramientas no menos interesantes (por orden alfabético) que incorpora GNU/Linux (p. ej. Debian) para la monitorización de sistema. No es una lista exhaustiva, sino una selección de las más utilizadas (se recomienda ver el *man* de cada herramienta para mayor información):

- atsar, ac, sac, sysstat, isag: herramientas de auditoría tales como ac, last, accton, sa, atsar o isag (Interactive System Activity Grapher) para la auditoría de recursos hw y sw.

- arpwatch; mon: monitor de actividad ethernet/FDDI que indica si hay cambios en tablas MACIP; monitor de servicios de red.

- diffmon, fcheck: generación de informes sobre cambios en la configuración del sistema y monitorización de los sistemas de ficheros para detectar intrusiones.

- fam: file alteration monitor.

- genpower: monitor para gestionar los fallos de alimentación.

- gkrellm: monitorización gráfica de CPU, procesos (memoria), sistemas de ficheros y usuarios, disco, red, Internet, *swap*, etc.

- ksensors: (lm-sensors): monitor de placa base (temperatura, alimentación, ventiladores, etc.)

- .lcap, systune: retira capacidades asignadas al *kernel* en el fichero /proc/sys/kernel y adapta según las necesidades con systune.

- logwatcher: analizador de logs.

- Munin y monit: monitorización gráfica del sistema.

- powertweak y gpowertweak: monitorización y modificación de diferentes parámetros del hardware, *kernel*, red, VFS, o VM (permite modificar algunos de los parámetros mostrados anteriormente sobre /proc).

- gps, gtop, tkps, lavaps (de más a menos amigable): monitores de procesos de varios tipos (generalmente utilizan información de /proc) y permiten ver recursos, *sockets*, archivos, entorno y otra información que éstos utilizan, así como administrar sus recursos/estados.

- *swatch*: monitor para la actividad del sistema a través de archivos de *log*.

- vtgrab: monitorización de máquinas remotas (similar a VNC).

- whowatch: herramienta en tiempo real para la monitorización de usuarios.

- wmnd, dmachinemon: monitor de tráfico de red y monitorización de un *cluster* por red.

- xosview, si: monitor de recursos gráfico y System Information.

La figura siguiente muestra las interfaces de ksensors, gkrellm y xosview, que presentan los resultados de la monitorización en tiempo real que están realizando.

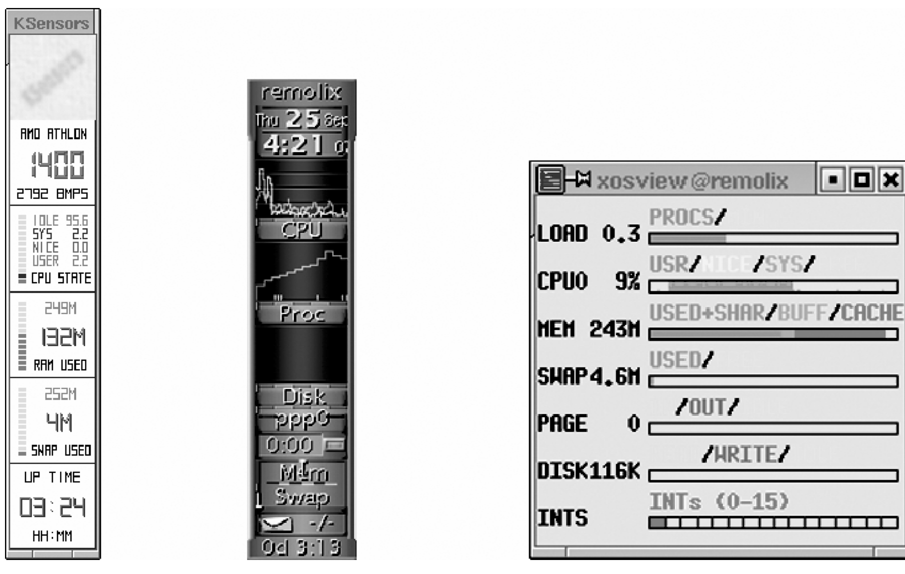


Figura 5

A continuación se muestran las interfaces gráficas de isag y gtop. La interfaz isag obtiene la información generada por systat en /etc/cron.d/, sysstat a través del comando sa1 y sa2 en este caso, y es el acumulativo del día; mientras que gtop muestra una de sus posibles vistas con la ubicación proceso, memoria e información complementaria de CPU.

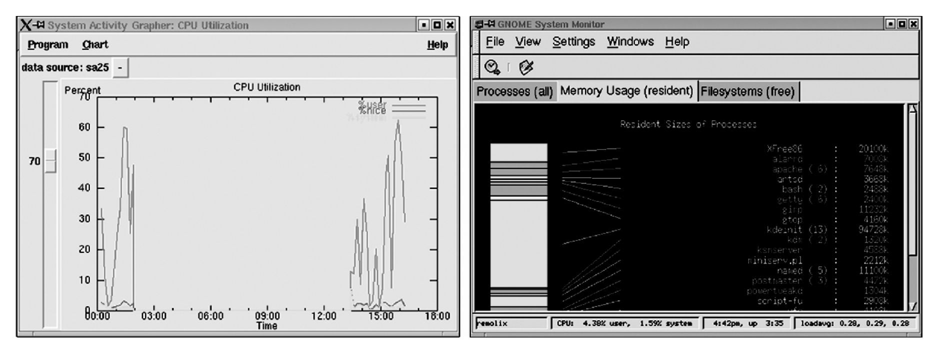


Figura 6

Actividades

1) Realizar una monitorización completa del sistema con las herramientas que consideréis adecuadas y hacer un diagnóstico de la utilización de recursos y cuello de botella que podrían existir en el sistema. Simular la carga en el sistema del código de sumdis.c dado en la unidad que trata acerca de los *clusters*. Por ejemplo, utilizar:

sumdis 1 2000000

2) Cambiar los parámetros del *kernel* y del compilador y ejecutar el código mencionado en el punto anterior (sumdis.c) con, por ejemplo:

time ./sumdis 1 1000000

3) También con ambos *kernels* y extraer conclusiones sobre los resultados.

Otras fuentes de referencia e información

[Debc, Ibi]

Optimización de servidores Linux: http://people.redhat.com/alikins/system_tuning.html
Performance Monitoring Tools for Linux http://www.linux

Munin: http://munin.projects.linpro.no/

Monit: http://www.tildeslash.com/monit/

Monitorización con Munin y monit:
http://www.howtoforge.com/server_monitoring_monit_munin

Monitorización con SNMP y MRTG: http://www.linuxhomenetworking.com/wiki/
index.php/Quick_HOWTO_:_Ch22_:_Monitoring_Server_Performance

MRTG: http://oss.oetiker.ch/mrtg/

Frysk: http://sources.redhat.com/frysk/

Cacti: http://cacti.net/